Data Structures

Indexed Data Structures - Indexed data structures are data structures that can be accessed through an index indicating the position of an element.

Summary The **Array** is mutable and fixed in size. It can be resized whenever an element is added past its initially allocated size, which requires copying to a separate array with a larger size. Resizing down to a smaller array also occurs in order to save space when elements are removed. In The resizing factors are determined by the programming language. The **String** is simply an array of characters, making it fixed in size, but can be resized depending on the language. Mutability is restricted to certain languages, but there are classes to go around mutability.

Runtimes Indexing Time: $O(1)$, Search Time: $O(n)$, Insertion Time at End: amortized $O(1)$. Insertion Time at Beginning or Middle: $O(n)$

Algorithms Linear search, two pointer approach, double pass from left and right, sliding window, dynamic programming on indices or growing array one at a time, divide and conquer.

Programming Variants

- Python does automatic resizing for its lists and Strings upon insertions and deletions.
- Java has an `ArrayList<T>` class that does automatic resizing of arrays and a `StringBuilder` that allows for concatenation of strings and mutability.
- C can contain a pointer that points to continuous bytes of the desired type, which makes it inherently a dynamic list as long as memory is allocated for the next set of bytes. Ex. `int *pointer = (int*) malloc(sizeof(int) * n);`. To have a fixed size but mutable in C, one can create a char array as `char arr[] = "code";`. A fixed size and immutable can be declared as a char pointer - `char *arr = "code";`

Hash Tables generate an ID for each key inserted and places into an array indexed by its hash

Summary The **Hash Table** generates a *roughly* unique ID. This allows for fast lookup but collisions may occur between keys that map to the same hash-value (hence roughly) via the algorithm because we only have a limited number of ”slots”. Collisions are represented by a linked list within the index by default (can also be binary trees and other data structures). A **Hash Set** has no value associated with its key, but a **Hash Map** does.

Runtimes Given we have m slots and n elements. Insertion, deletion, and lookup Best/Worst/Average: $O(1)/O(n)/O(\frac{n}{m})$.

Algorithms Accumulating key counts, keeping a set of ”seen so far” elements, adjacency lists.

Types and Variants

- Python has the `set()` operator that build a HashSet and a dictionary is the equivalent of a Hash Map initilized by `map = {}`.
- Java has interfaces `Set<T>` and `Map<T>` that are implemented by the `HashSet<T>` and `HashMap<T>` classes.

Linked List - Structure holding information and a pointer to the next structure of data in the list

Summary The basic **Singly Linked List** just has a single pointer to the next element and the last element has a pointer to NULL. Advantage over arrays is no fixed size, can be increased dynamically. Search cost is linear unless supplemented with a hashmap of keys to nodes (typical in an LRU cache).

Runtimes Insertion/Deletion: $O(n)$. Deletion and Insertion to the end or beginning (doubly linked or pointer to last element): $O(1)$. With a `HashMap<T>` of the key to the respective node, insertion and deletion from the middle is also $O(1)$ (but with more memory) - Commonly used in LRU caches

Algorithms Runner Pointer for list, moving pointers for insertion and deletion

Types and Variants

- Circular Linked List - the last element has a pointer to the beginning (usually a sentinal node that does not contain any information).
- Doubly Linked List - has two pointers, one to the element after it, and one to the element before. The element before the first element is the last element.

Binary Trees - Hierarchical structures where each node has ≤ 2 children and do not contain cycles

Summary -

Runtimes -

Algorithms -

Types and Variants

- Red-Black Trees

Heaps

Summary -

Runtimes -

Algorithms -

Types and Variants

- Red-Black Trees

Summary -

Runtimes -

Algorithms -

Types and Variants

- Red-Black Trees

Graphs

Summary -

Runtimes -

Algorithms -

Types and Variants

- Red-Black Trees

Stacks and Queues

Summary -

Runtimes -

Algorithms -

Types and Variants

- Red-Black Trees

Disjoint Sets

Summary -

Runtimes -

Algorithms -